METHOD FOR TESTING FLASH MEMORY POWER LOSS RECOVERY

TECHNICAL FIELD OF THE INVENTION

[0001] The present invention relates generally to non-volatile memory devices and in particular the present invention relates to Flash memory device management software, drivers, and power loss recovery testing.

BACKGROUND OF THE INVENTION

[0002] Memory devices are typically provided as internal storage areas in a computer. The term memory identifies data storage that comes in the form of integrated circuit chips. There are several different types of memory used in modern electronics, one common type is RAM (random-access memory). RAM is characteristically found in use as main memory in a computer environment. RAM refers to read and write memory; that is, you can both write data into RAM and read data from RAM. This is in contrast to ROM (read-only memory), which permits you only to read data. Most RAM is volatile, which means that it requires a steady flow of electricity to maintain its contents. As soon as the power is turned off, whatever data was in RAM is lost.

[0003] Computers almost always contain a small amount of ROM that holds instructions for starting up the computer, typically called a basic input output system (BIOS). Unlike RAM, ROM generally cannot be written to by a user. An EEPROM (electrically erasable programmable read-only memory) is a special type non-volatile ROM that can be erased by exposing it to an electrical charge. EEPROM comprise a large number of memory cells having electrically isolated gates (floating gates). Data is stored in the memory cells in the form of charge on the floating gates. Charge is transported to or removed from the floating gates by specialized programming and erase operations, respectively.

[0004] Yet another type of non-volatile memory is a Flash memory. A Flash memory is a type of EEPROM that can be erased in blocks instead of one byte at a time. A typical Flash memory comprises a memory array, which includes a large number of memory cells.

The memory cells of a Flash memory array are typically arranged into a "NOR" architecture (where each cell directly coupled to a bitline) or a "NAND" architecture (where the cells are coupled into "strings" of cells, such that each cell is coupled indirectly to a bitline and requires activation of the other cells of the string for access to a selected cell). Each of the memory cells includes a floating gate field-effect transistor capable of holding a charge. The data in a cell is determined by the presence or absence of the charge in the floating gate. The cells are usually grouped into sections called "erase blocks." Each of the cells within an erase block can be electrically programmed in a random basis by charging the floating gate. The charge can be removed from the floating gate by a block erase operation, wherein all floating gate memory cells in the erase block are erased in a single operation. Other types of non-volatile memory include, but are not limited to, Polymer Memory, Ferroelectric Random Access Memory (FeRAM), Ovionics Unified Memory (OUM), and Magnetoresistive Random Access Memory (MRAM).

[0005] Many Flash memory devices are utilized with specialized software handling and/or management routines, generally referred to as "drivers." The Flash memory drivers are executed on the "host," typically a processor or memory controller, and allow the Flash memory device(s) being utilized to be read from and written to by the host. The drivers also provide a layer of logical abstraction for the host; presenting the Flash memory device as a freely re-writeable general access memory device or mass storage device, such as a hard drive, a floppy disk, or other non-volatile machine-usable storage device. The drivers, as part of the Flash memory device software interface/hardware abstraction, typically also manage the internal operation of the Flash memory device; scheduling erase blocks to be erased, managing bad erase blocks, protecting and unprotecting erase blocks, power loss recovery, and load leveling (also called wear leveling) the Flash memory device.

[0006] The driver and/or memory management routines are generally supplied by the Flash memory manufacturer to the end-user or system manufacturer. These driver routines are typically supplied in a source code format or as a linkable library and as such must be compiled into the operating system or overall code executing on the device. Self

contained and separately loadable drivers also exist, but are typically not utilized in embedded processor devices.

[0007] A problem with utilizing Flash memory devices, particularly in embedded processor or portable battery powered systems, is dealing with potentially intermittent power sources or power availability. As Flash memory is non-volatile, once data is stored it will not be affected by loss of power. However, as data write operations, block erasures, and other internal Flash management operations that store or change data or the operating state of the Flash memory device are generally not single step operations, the Flash memory can be left in an incomplete state by a power loss event. For example, a make directory command requires 43 separate write/erase operations by the driver of a specific Flash memory device to complete. Because of these issues the driver software contains "power loss recovery" routines that sequence the Flash memory device through a power loss recovery cycle to check for such uncompleted operations upon power up, and, if possible, finish or correct them.

[0008] The testing of these power loss recovery cycles and drivers routines (herein referred to as the power loss recovery cycle) in prior art Flash memory devices and systems has generally involved randomized power loss testing. In this form of system testing, power is removed from the system at a random point during a selected write, erase, or Flash memory management operation and then is restored, testing the power loss recovery cycle and associated driver routines. The testing of a selected write, erase, or Flash memory management operation is continued in this manner for an indefinite period of time (as much as 2 to 3 days), until an appropriate level of confidence in the power loss recovery cycle is achieved. This approach is problematic in that it is empirically based and cannot completely guarantee even after a large number of testing cycles that an error is not present in the power loss recovery cycle or routines; the true coverage of the test is unknown. This form of testing is also time consuming and requires exhaustive testing or retesting when a section of code or hardware is written or changed to verify its correct operation.

[0009] For the reasons stated above, and for other reasons stated below which will become apparent to those skilled in the art upon reading and understanding the present specification, there is a need in the art for alternative apparatus and methods of testing Flash memory devices, their driver programs, and power loss recovery cycles.

SUMMARY

[0010] The above-mentioned problems with testing Flash memories, Flash memory drivers, power loss recovery testing, and other problems are addressed by the present invention and will be understood by reading and studying the following specification.

[0011] The various embodiments of the present invention relate to deterministic testing and verification of non-volatile memories, non-volatile memory drivers, write operations/cycles, block erase operations/cycles, non-volatile memory management operations, and power loss recovery testing. The memory device and software embodiments of the present invention utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point. This ability to utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point allows for halting or interrupting execution at each point in a selected operation that a non-volatile memory cell is programmed or erased. This allows for a deterministic and repeatable testing process of all stages of a write cycle or erase cycle where the state of non-volatile memory cells are changed in the non-volatile memory device. This also allows for step-through testing of each stage by programmer/designer of state changes non-volatile memory cells in the non-volatile memory device or driver software, enabling quick testing and easy logic and code changes, saving time and resources. Additionally, this ability to utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point enables simulation of power loss at each point in a selected operation that a non-volatile memory cell is programmed or erased, allowing for improved, deterministic testing of the power loss recovery cycle and faster code/design change verification.

[0012] For one embodiment, the invention provides a method of operating a non-volatile memory device driver comprising counting a number of access cycles to a non-volatile memory, and halting execution at a selected count.

[0013] For another embodiment, the invention provides a method of operating a system comprising counting a number of access operations to a Flash memory device coupled to a host, and stopping execution at a selected number of access operations.

[0014] For yet another embodiment, the invention provides a method of testing a Flash memory comprising counting a number of access operations to a Flash memory for a Flash command, interrupting execution of the Flash command at a selected halt count of access operations, and executing a power loss recovery cycle to test power loss recovery at the selected halt count.

[0015] For a further embodiment, the invention provides a system comprising at least one Flash memory device, and a host coupled to the at least one Flash memory device, wherein the host is adapted to count a number of access operations to the at least one Flash memory device during a Flash command and halt execution of the Flash command at a selected count of access operations.

[0016] Further embodiments of the invention include methods and apparatus of varying scope.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] Figure 1 is a simplified block diagram of a system containing a Flash memory device in accordance with an embodiment of the present invention.

[0018] Figure 2 is a simplified block diagram of a system containing a Flash memory subsystem in accordance with an embodiment of the present invention.

[0019] Figures 3A and 3B are simplified flowchart diagrams detailing operation of methods in accordance with embodiments of the present invention.

DETAILED DESCRIPTION

[0020] In the following detailed description of the invention, reference is made to the accompanying drawings that form a part hereof, and in which is shown, by way of illustration, specific embodiments in which the invention may be practiced. In the drawings, like numerals describe substantially similar components throughout the several views. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, and electrical changes may be made without departing from the scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims and equivalents thereof.

[0021] Embodiments of the present invention include non-volatile memory drivers, testing methods, and apparatuses that allow for deterministic testing and verification of non-volatile memories, non-volatile memory drivers, write operations/cycles, block erase operations/cycles, non-volatile memory management operations, and power loss recovery testing. In one embodiment, this accomplished by a counter in a low level driver that counts the number of program/erase operations or cycles. The memory device and software embodiments of the present invention utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point. This ability to utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point allows for halting or interrupting execution of the driver/software routine at each point that a non-volatile memory cell is programmed or erased in a selected operation. This allows for a deterministic and repeatable testing process of all stages of a write cycle or erase cycle where the state of non-volatile memory cells are changed in the non-volatile memory device. This also allows for step-through testing of each stage by programmer/designer of state changes of the non-volatile memory cells in the non-volatile memory device or driver software, enabling quick testing and easy logic and code changes, saving time and resources. Additionally, this ability to utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point enables simulation of power loss at each point in a selected

operation that a non-volatile memory cell is programmed or erased, allowing for improved, deterministic testing of the power loss recovery cycle and faster code/design change verification.

[0022] Because all the cells in an erase block of a Flash memory device must be erased at once, one cannot directly rewrite a Flash memory cell without first engaging in a block erase operation. Erase block management (EBM), typically a function of the driver, provides an abstraction layer for this to the host (a processor or an external memory controller), allowing the Flash device to appear as a freely rewriteable device, including, but not limited to, managing the logical address to physical erase block translation mapping for reads and writes, the assignment of erased and available erase blocks for utilization, and the scheduling erase blocks that have been used and closed out for block erasure. Data objects, which can be individual sectors, erase blocks, or logical file system entities, are created and managed through creation, updating, and invalidation/erasure by the EBM. Erase block management also allows for load leveling of the internal floating gate memory cells to help prevent write fatigue failure. Write fatigue is where the floating gate memory cell, after repetitive writes and erasures, no longer properly erases and removes charge from the floating gate. Load leveling procedures increase the mean time between failure of the erase block and Flash memory device as a whole.

[0023] The complexity of the tasks of managing and interfacing to the Flash memory device(s) are such that the driver software/memory management software is typically segmented into a data manager layer (that is responsible for the higher level interfacing such as erase block management and address/logical device abstraction) and a low level device driver layer (that is responsible for the interfacing, command set sequences, and timing of interfacing to the specific memory device and provides basic read/program/erase functionality). These driver software/memory management software layers typically interface between each other by means of a defined "application interface" (API) that allows the differing layers to function without direct knowledge or control of the other layers.

[0024] Additionally, there is in some cases a further driver layer between the operating system/application, called the file manager. The file manager is responsible for managing filesystem data entities (typically separate data files or filesystem structures) and the logical structure/filesystem format of the memory device. The file manager (or in some cases, the data manager) can tailor its operation of the memory device to the device's storage usage by the operating system/application. The storage usage or access patterns of the memory device are called the "data model" of the memory use and can be used to optimize the memory's utilization by the application/system it is installed in. For example, a digital camera would require large data file storage that was frequently erased and reprogrammed, whereas a cell phone would typically require storage of many small data entities that would be frequently accessed but seldom changed.

[0025] After the overall driver's application interface (API) to the application/operating system acknowledges the write or erase operation request the successful execution of the data write/erasure is guaranteed. In executing these requests, a driver can operate in synchronous or asynchronous operating mode dependant on when it acknowledges the Flash operation request to the operating system/application. If the driver is still in the process of completing execution of the operation on the Flash memory device when it acknowledges the write or erase operation, the driver is known as an "asynchronous" driver and typically only occurs in systems capable of multi-tasking. This is opposed to "synchronous" driver operation where execution of the main operating system/application halts until the driver completes the entire operation.

[0026] The software routines and drivers that operate computer based devices are sometimes referred to as firmware or ROM after the non-volatile ROM machine-usable storage device that such routines have historically been stored in. It is noted that firmware or software routines can be stored on a variety of machine-usable storage mediums or firmware storage mediums that include, but are not limited to, a non-volatile Flash memory, a ROM, an EEPROM, a one time programmable (OTP) device, a complex programmable logic device (CPLD), an application specific integrated circuit (ASIC), a magnetic media disk, etc.

[0027] In many modern Flash memory devices implementations, the host interface and erase block management routines additionally allow the Flash memory device to appear as a read/write mass storage device (i.e., a magnetic disk) to the host. One such approach is to conform the interface to the Flash memory to be identical to a standard interface for a conventional magnetic hard disk drive allowing the Flash memory device to appear as a block read/write mass storage device or disk. This approach has been codified by the Personal Computer Memory Card International Association (PCMCIA), Compact Flash (CF), and Multimedia Card (MMC) standardization committees, which have each promulgated a standard for supporting Flash memory systems or Flash memory "cards" with a hard disk drive protocol. A Flash memory device or Flash memory card (including one or more Flash memory array chips) whose interface meets these standards can be plugged into a host system having a standard DOS (Disk Operating System) or compatible operating system with a Personal Computer Memory Card International Association – Advanced Technology Attachment (PCMCIA-ATA) or standard ATA interface. Other additional Flash memory based mass storage devices of differing low level formats and interfaces also exist, such as Universal Serial Bus (USB) Flash drives, Secure Digital Memory Card, or Sony MemoryStick.

[0028] Many of the modern computer operating systems, such as DOS, were developed to support the physical characteristics of hard drive structures; supporting file structures based on heads, cylinders and sectors. The DOS software stores and retrieves data based on these physical attributes. Magnetic hard disk drives operate by storing polarities on magnetic material. This material is able to be rewritten quickly and as often as desired. These characteristics have allowed DOS to develop a file structure that stores files at a given location which is updated by a rewrite of that location as information is changed. Essentially all locations in DOS are viewed as fixed and do not change over the life of the disk drive being used therewith, and are easily updated by rewrites of the smallest supported block of this structure. A sector (of a magnetic disk drive) is the smallest unit of storage that the DOS operating system supports. In particular, a sector has come to mean 512 bytes of information for DOS and most other operating systems in existence. Flash memory systems that emulate the storage characteristics of hard disk

drives are preferably structured to support storage in 512 byte blocks along with additional storage for overhead associated with mass storage, such as ECC (error correction code) bits, status flags for the sector or erase block, and/or redundant bits.

[0029] Figure 1 shows a simplified diagram of a system 128 incorporating a Flash memory device 100 and driver of the present invention coupled to a host 102, which is typically a processing device or memory controller. The Flash memory device 100 has an address interface 104, a control interface 106, and a data interface 108 that are each coupled to the processing device 102 to allow memory read and write accesses. Internal to the Flash memory device, a control state machine 110 directs the internal operation; managing the Flash memory array 112 and updating RAM control registers and non-volatile erase block management registers 114. The RAM control registers and tables 114 are utilized by the control state machine 110 during operation of the Flash memory device 100. The Flash memory array 112 contains a sequence of memory banks or segments 116. Each bank 116 is organized logically into a series of erase blocks (not shown). Memory access addresses are received on the address interface 104 of the Flash memory device 100 and divided into a row and column address portions. On a read access the row address is latched and decoded by row decode circuit 120, which selects and activates a row page (not shown) of memory cells across a selected memory bank. The bit values encoded in the output of the selected row of memory cells are coupled from a local bitline (not shown) to a global bitline (not shown) and detected by sense amplifiers 122 associated with the memory bank. The column address of the access is latched and decoded by the column decode circuit 124. The output of the column decode circuit 124 selects the desired column data from the internal data bus (not shown) that is coupled to the outputs of the individual read sense amplifiers 122 and couples them to the data buffer 126 for transfer from the memory device 100 through the data interface 108. On a write access the row decode circuit 120 selects the row page and column decode circuit 124 selects write sense amplifiers 122. Data values to be written are coupled from the data buffer 126 via the internal data bus to the write sense amplifiers 122 selected by the column decode circuit 124 and written to the selected floating gate memory cells (not shown) of the memory array 112. The written cells are then reselected by the row and column decode circuits

120, 124 and sense amplifiers 122 so that they can be read to verify that the correct values have been programmed into the selected memory cells.

[0030] Figure 2 is a simplified diagram of another system 200 that incorporates a Flash memory subsystem 204 and associated Flash driver software routines of an embodiment of the present invention. In the system 200 of Figure 2, the Flash memory subsystem 204, such as a memory system or Flash memory card, is coupled to a processor 202 with a synchronous interface having an address 206, control 208, and data bus 210. Internal to the Flash memory system 204, a memory controller 212 directs internal operation of the Flash memory system 204; managing the Flash memory devices 214, directing data accesses, updating internal control registers and tables (not shown), and/or directing operation of other possible subsystems (not shown) of the Flash memory system 204. The memory controller 212 is coupled to and controls one or more Flash memory devices 214 via an internal bus 236. Each Flash memory device 214 contains a sequence of erase blocks (not shown) in an internal memory array 216. It is noted that other architectures of Flash memory systems 204, external interfaces 206, 208, 210, and manners of coupling the memory controller 212 to the Flash memory devices 214, such as directly coupled individual control busses and signal lines, are possible and should be apparent to those skilled in the art with benefit of the present disclosure.

[0031] On power up of a system incorporating a Flash memory device or system, the driver typically executes the power loss recovery routine and scans the entire Flash memory, examining each data object by checking its status in an EBM data field. If the data object is marked in the EBM field as "complete" it is left alone. If the object is marked as "being updated", the object is checked and, if possible, the remaining steps to place the data object in a "complete" state are executed. If the data object is unable to be completed it is marked as "invalid." In the segmented driver described above, power loss recovery is generally the data manager's task to scan the Flash memory and correct or discard incomplete data objects, although in some cases the file manager, if present in the driver, may also be involved.

[0032] In embodiments of the present invention, the driver counts the number of write or erase operations for executing a selected Flash operation and then halts/interrupts execution at a selected write or erase operation. At this point the state of the processor and/or contents of the memory and registers can be inspected if desired. If the system is then brought back up again this tests the power loss recovery for the selected halt point in the Flash command and allows a known and definite testing coverage of the Flash memory device and driver for a particular case. This quantifies the power loss case for that particular software environment (and if desired, physical environment). In one embodiment, the counting of write or erase operations and/or halting/interrupting execution is done by a function incorporated into the low level driver. In one embodiment only write or only erase operations are counted. In another operation both write and erase operations are counted.

[0033] Write and erase commands are not executed in a rapid manner in typical Flash memory devices and can take up to several milliseconds to complete. This write/erase command time period is typically counted in the multiples of clock cycles for most hosts (i.e., processors or memory controllers). As such, in one embodiment of the present invention a driver executing on a host counts the number of write and/or erase operations for executing a selected Flash operation and at a selected write or erase operation counts (typically on a counter internal to the host) the number of clock cycles/time period into the execution of the write or erase command by the Flash memory device before halting/interrupting execution of the selected write or erase command. This allows the embodiment to simulate actual power loss at a given point in execution of a write/erase command and/or command sequence of a Flash memory device. This is particularly advantageous for testing Flash memory devices with multi-level cells, where each memory cell contains two or more memory data bits which are typically programmed in a single write operation. In another embodiment, as numerous Flash memory devices do not have a method of halting the execution of Flash commands that are executing internal to the device, this halting of execution is accomplished by the host triggering external hardware to remove power from the Flash memory device under test at a selected time or clock cycle count after issuing a Flash write or erase command.

[0034] In another embodiment of the present invention, the non-volatile memory device has a special purpose internal register that may be set to halt or stop the internal execution of a command in its command execution logic after a selected number of system clock cycles, or, in the case of a command execution logic state machine, a selected number of state machine steps. In one embodiment, the driver would load the special purpose internal register of the non-volatile memory device to halt execution of a command at a selected number of system clocks or command execution state machine steps and then loads and executes a command or command sequence to test. The non-volatile memory executes the command and halts/interrupts command execution at the selected number of system clocks or command execution state machine steps.

[0035] Figure 3A shows a simplified flowchart 300 to a method of operating a Flash memory device driver of an embodiment of the present invention. In Figure 3A, the memory device driver counts the number of erase and write operations 302 done to the Flash memory. At a selected count, the device driver then halts execution 304 of the current Flash memory command. At this point the state of the processor and/or content of the memory and registers of the system executing the device driver can optionally be inspected. The device driver then reboots/brings back up the system and the power loss recovery routine is tested 306 for the Flash memory command that was executed. If the halt is before the point where the Flash memory command can be completed by the power loss recovery routine, the data object should be placed/marked as invalid. If the sequence of writes and/or erases is halted after the Flash memory command can be completed, the power loss recovery routine should finish the execution of the command and mark the data object into a completed state.

[0036] With the addition of a step to automatically increment the write/erase count where the command is halted, the method of operating a device driver of Figure 3A implements an iterative test of the executed Flash memory command. This allows a quantification of the power loss case for the selected Flash memory command. For example, incrementing through the above mentioned 43-step Flash operation to execute a

make directory command allows the make directory command of the driver to be deterministically tested.

[0037] Figure 3B shows a simplified flowchart 320 to a method of operating a Flash memory device driver of another embodiment of the present invention. In Figure 3B, the memory device driver counts the number (N) of erase and write operations 322 done to the Flash memory. Starting from a selected halt count of one, the device driver then halts execution 324 of the current Flash memory command. The device driver then reboots/brings back up the system and the power loss recovery routine is tested 326 for the Flash memory command that was executed. After the power loss recovery is tested, the selected halt count is incremented 328 and the method loops 330 to repeat the Flash command halting now at the new incremented halt count. The method then repeats incrementing the halt count, halting on each repetition at a new halt count (i.e., 1, 2, 3, . . , N), until power loss recovery has been tested by halting on each write/erase operation of the Flash memory command being tested. This allows a fast and definite testing coverage of the Flash memory device and driver for the selected Flash command, allowing a quantification of the power loss case for the selected command.

[0038] If the driver of Figure 3B is utilized to step through and test each command of a Flash memory device command set, an iterative test of a whole command set is executed. This allows the user to test all known power loss cases of a Flash memory command set in a fast and deterministic manner. The testing of the write and erase operations of a selected Flash memory device and driver can be accomplished in a smaller period of time and with a high level of confidence in the power loss recovery cycle; helping to ensure that a latent error is not hiding in the power loss recovery cycle or routines. This deterministic driver testing also allows quick iterations of driver code development and verification of the code operation.

[0039] It is noted that other testing patterns and methodologies, such as starting with the final write/erase operation of the command and decrementing or verification of block management and erase commands before write commands, are possible for Flash memory

command testing with embodiments of the present invention and will be apparent to those skilled in the art with the benefit of the present disclosure.

[0040] In another embodiment of the present invention, the write/erase counter of the driver allows for profiling of the number of write/erase operations a command utilizes in its execution by counting the number of write/erase operations it takes to complete. This profiling of a command can allow for automatic testing by allowing test run time determination of the number of write/erase operations of a command. Profiling can also allow the user to see what, if anything, has changed in the implementation of a command. This enables easy comparison of code versions, different command versions, or code from a separate source (i.e., a customer modified version of the code that was sent back to manufacturer for debugging and analysis).

[0041] In another embodiment of the present invention, the write/erase counter of the driver allows for code debugging by allowing the code to be stopped and the state of the processor and memory to be examined whenever information is recorded or erased in the Flash. This is particularly useful in debugging intermittent errors in the operation of the Flash memory. Differing inputs and environment stresses (such as voltage, temperature, noise, etc.) may be applied to the system or Flash memory and write or erase operations repeatedly run to locate problematic conditions or errors.

[0042] It is noted that driver testing with other embodiments of the present invention are possible and should be apparent to those skilled in the art with the benefit of the present disclosure.

CONCLUSION

[0043] A non-volatile memory device, driver, and method has been described that utilizes write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point to interrupt/stop execution or simulate power loss at a specific point. The various embodiments of the present invention relate to deterministic testing and verification of non-volatile memories, non-volatile memory drivers, write operations/cycles, block erase operations/cycles, non-volatile memory

management operations, and power loss recovery testing. The memory device and software embodiments of the present invention utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point. This ability to utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at a selected point allows for halting or interrupting execution at each point in a selected operation that a non-volatile memory cell is programmed or erased. This allows for a deterministic and repeatable testing process of all stages of a write cycle or erase cycle where the state of non-volatile memory cells are changed in the non-volatile memory device. This also allows for step-through testing of each stage by programmer/designer of state changes non-volatile memory cells in the non-volatile memory device or driver software, enabling quick testing and easy logic and code changes, saving time and resources. Additionally, this ability to utilize write or erase cycle tracking to interrupt or stop a non-volatile memory programming operation at any selected point enables simulation of power loss at each point in a selected operation that a non-volatile memory cell is programmed or erased, allowing for improved, deterministic testing of the power loss recovery cycle and faster code/design change verification.

[0044] Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement that is calculated to achieve the same purpose may be substituted for the specific embodiments shown. Many adaptations of the invention will be apparent to those of ordinary skill in the art. Accordingly, this application is intended to cover any adaptations or variations of the invention. It is manifestly intended that this invention be limited only by the following claims and equivalents thereof.